Fast Cryptography in Genus 2

Joppe W. Bos

Joint work with

Craig Costello, Huseyin Hisil, Kristin Lauter

Workshop on Elliptic Curve Cryptography 2013

Microsoft Research

Fast Cryptography in Genus 2

From a practical perspective!

Joppe W. Bos

Joint work with

Craig Costello, Huseyin Hisil, Kristin Lauter

Workshop on Elliptic Curve Cryptography 2013

Microsoft Research

Motivation - I

This is the ECC Workshop: we all like (elliptic) curves!

	DH	ECDH
Group	$(\mathbf{F}_{p_1}^*, \times)$	$(E(\mathbf{F}_{p_2}), +)$

Motivation - I

This is the ECC Workshop: we all like (elliptic) curves!

	DH	ECDH
Group	$(\mathbf{F}_{p_1}^*, \times)$	$(E(\mathbf{F}_{p_2}), +)$

Why?

Size of p!

Security level (bits)	$\log_2 p_1$	$\log_2 p_2$
128	3072	256
192	7680	384
256	15360	521

Source: NSA – The case for Elliptic Curve Cryptography
http://www.nsa.gov/business/programs/elliptic_curve.shtml

Motivation - I

This is the ECC Workshop: we all like (elliptic) curves!

	DH	ECDH
Group	$(\mathbf{F}_{p_1}^*, \times)$	$(E(\mathbf{F}_{p_2}), +)$

Why?

Performance!

Security level (bits)	$\log_2 p_1$	$\log_2 p_2$	Ratio DH cost : ECDH cost
128	3072	256	10:1
192	7680	384	32:1
256	15360	521	64:1

Source: NSA – The case for Elliptic Curve Cryptography
http://www.nsa.gov/business/programs/elliptic_curve.shtml

Motivation - I

This is the ECC Workshop: we all like (elliptic) curves!

	DH	ECDH
Group	$(\mathbf{F}_{p_1}^*, \times)$	$(E(\mathbf{F}_{p_2}), +)$

Why?

Performance!

Security level (bits)	$\log_2 p_1$	$\log_2 p_2$	Ratio DH cost : ECDH cost
128	3072	256	10:1
192	7680	384	32:1
256	15360	521	64:1

Source: NSA – The case for Elliptic Curve Cryptography
http://www.nsa.gov/business/programs/elliptic_curve.shtml

Can we do better?

Can we do better?

Reduce the **cost** of the group operation

- Use a different curve representation
- Use a different coordinate system
- E.g. **twisted Edwards curves** with **extended twisted Edwards coordinates**
- See the Explicit-Formulas Database

Can we do better?

Reduce the **cost** of the group operation

- Use a different curve representation
- Use a different coordinate system
- E.g. **twisted Edwards curves** with **extended twisted Edwards coordinates**
- See the Explicit-Formulas Database

Reduce the **number** of group operations

- Reduce the number of **point additions**
  e.g. use large window sizes
- Reduce the number of **point doublings**
  e.g. scalar decomposition

Can we do better?

Reduce the **cost** of the group operation

- Use a different curve representation
- Use a different coordinate system
- E.g. **twisted Edwards curves** with **extended twisted Edwards coordinates**
- See the Explicit-Formulas Database

Reduce the **number** of group operations

- Reduce the number of **point additions**
  e.g. use large window sizes
- Reduce the number of **point doublings**
  e.g. scalar decomposition

Other optimizations

- Montgomery ladder
- Fast finite field arithmetic:
  Curves over “special” primes
- Implementations using all the features of the architecture: e.g. special instructions, SIMD instructions

Can we do better?

Reduce the **cost** of the group operation

- Use a different curve representation
- Use a different coordinate system
- E.g. **twisted Edwards curves** with **extended twisted Edwards coordinates**
- See the Explicit-Formulas Database

Reduce the **number** of group operations

- Reduce the number of **point additions**
  e.g. use large window sizes
- Reduce the number of **point doublings**
  e.g. scalar decomposition

Other optimizations

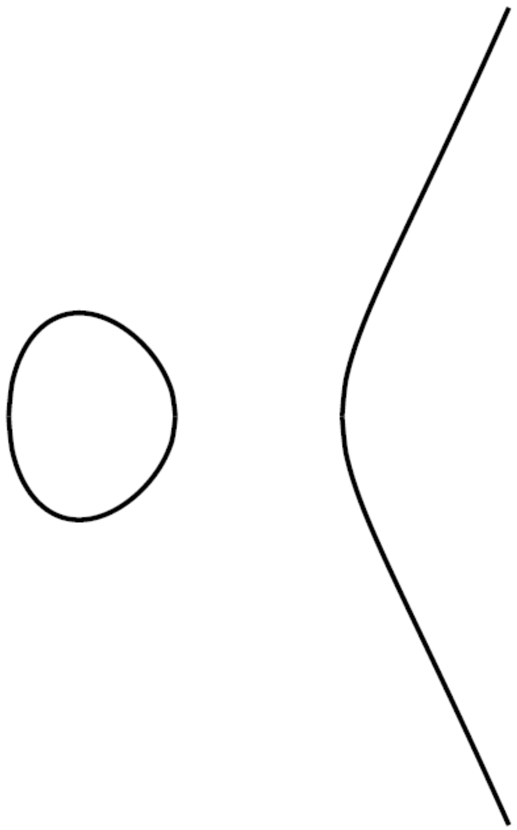
- Montgomery ladder
- Fast finite field arithmetic: Curves over “special” primes
- Implementations using all the features of the architecture: e.g. special instructions, SIMD instructions

Change the setting!

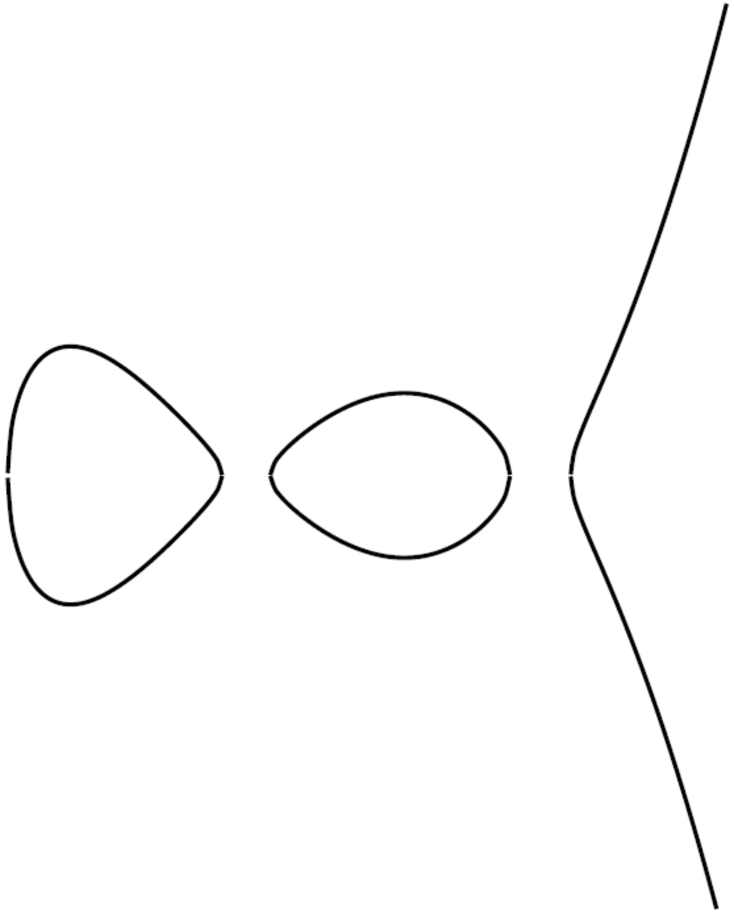
- Consider genus 2
  - Different cost of the group operation
  - Different number of group operations
- Genus 2 equivalent of Montgomery ladder
  - Kummer surface
- GLV on genus 2 curves?

Why genus 2?

$y^2 = x^3 + a_2x^2 + a_1x + a_0$



$y^2 = x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

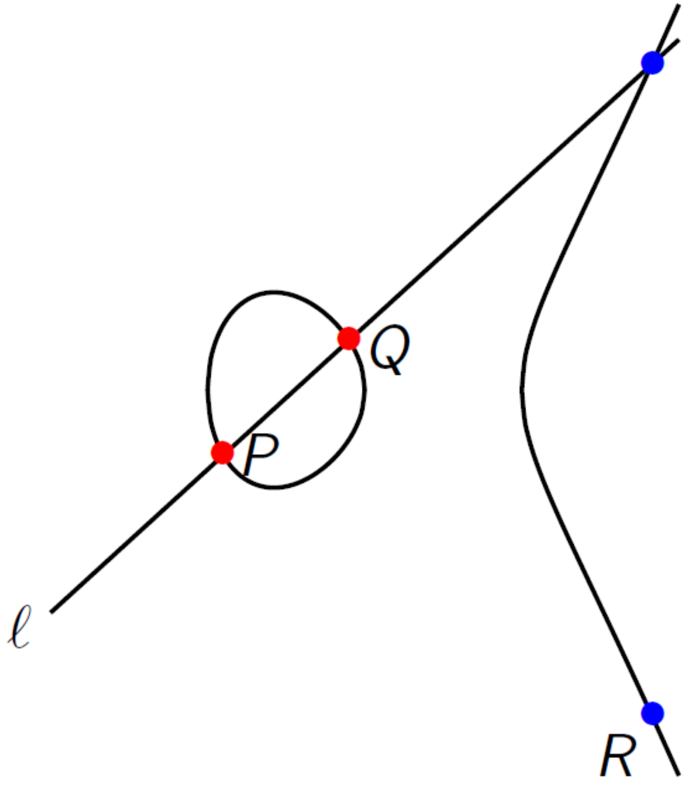


Both curves have around p points over $\mathbf{F}_p$

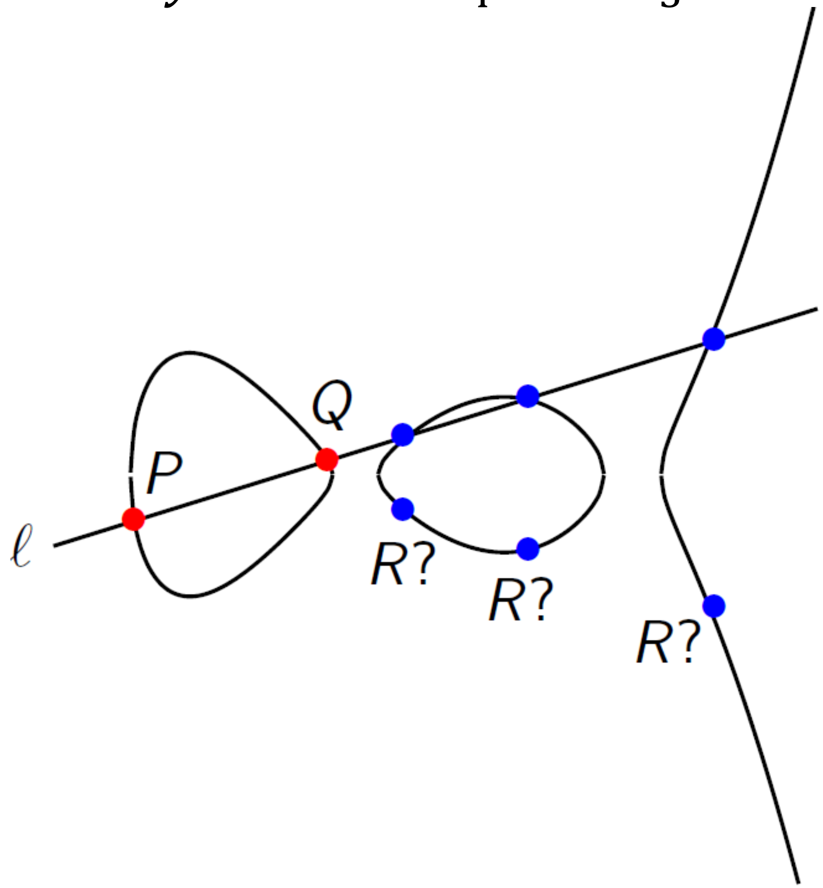
Hasse-Weil: $p + 1 - 2g\sqrt{p} \leq \#C(\mathbf{F}_p) \leq p + 1 + 2g\sqrt{p}$

Why genus 2?

$y^2 = x^3 + a_2 x^2 + a_1 x + a_0$



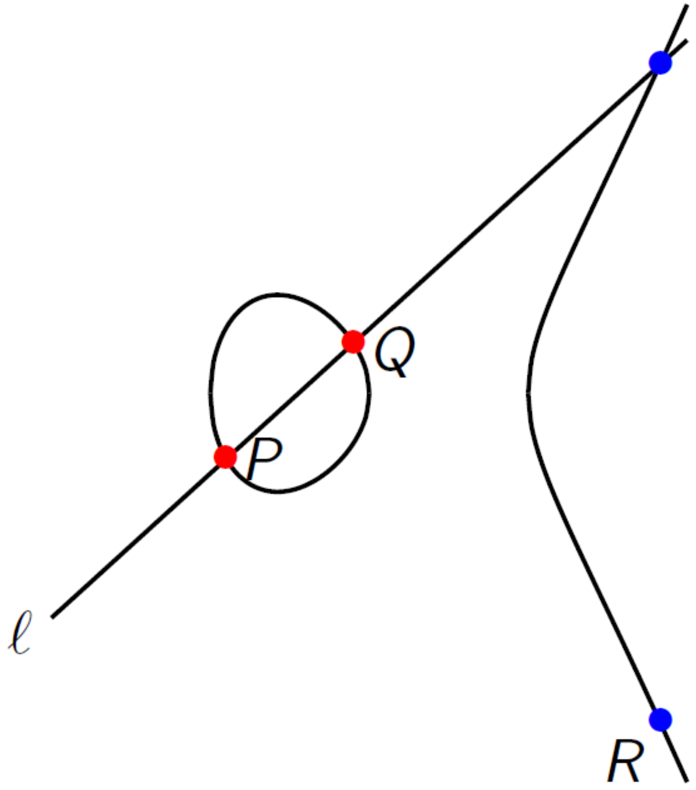
$y^2 = x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$



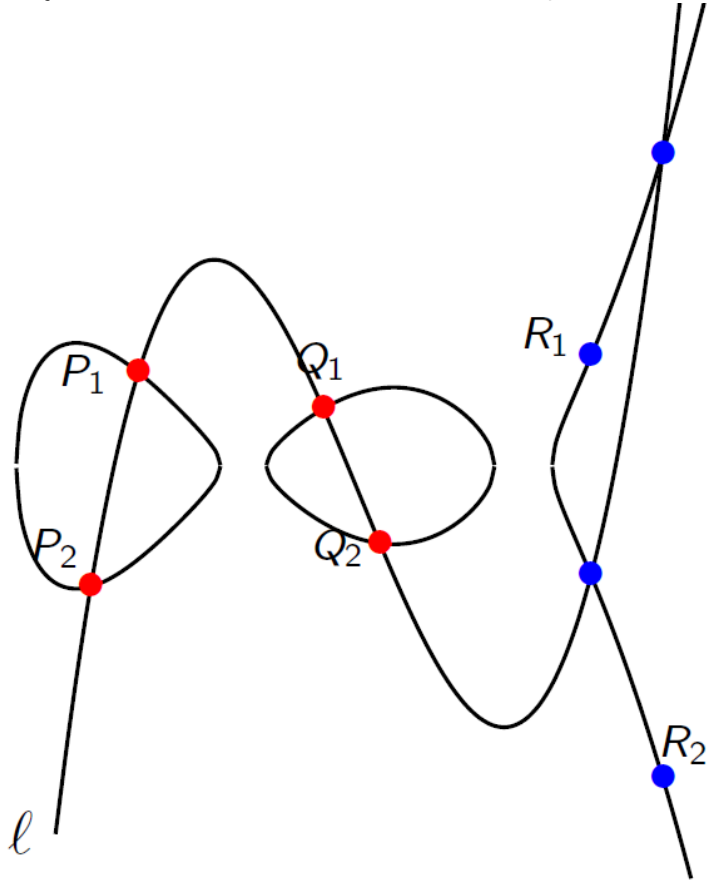
Can’t do “chord-and-tangent” in genus 2

Why genus 2?

$y^2 = x^3 + a_2x^2 + a_1x + a_0$



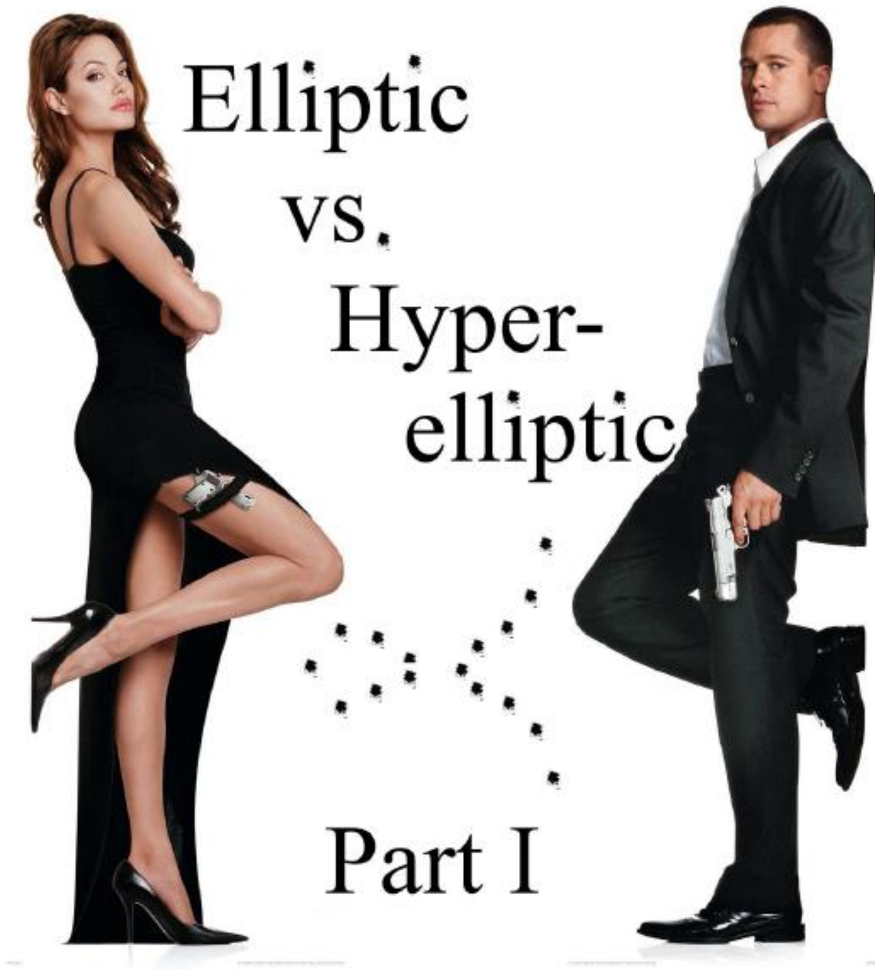
$y^2 = x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$



Roughly speaking: group elements are pairs of points

$\#E(\mathbf{F}_p) \approx p$ versus $\#\mathrm{Jac}_C(\mathbf{F}_p) \approx p^2$

Flashback?!

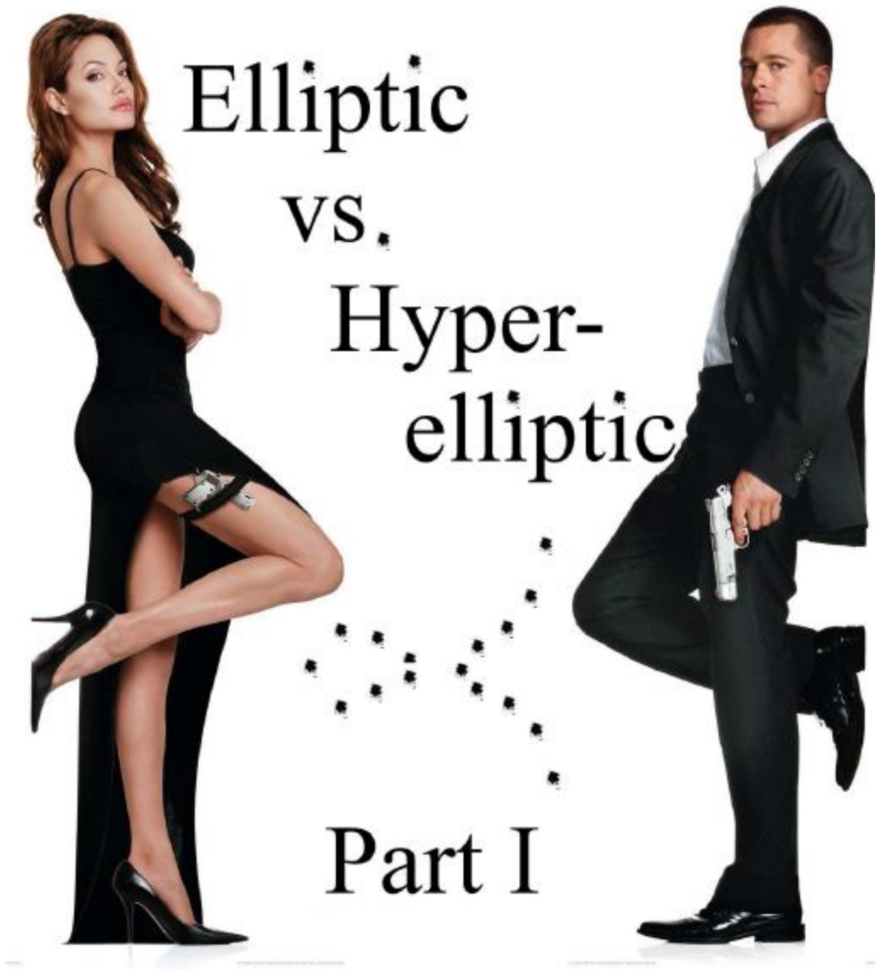


Wasn't this considered before?

2006: D. J. Bernstein: *Elliptic vs. hyperelliptic*, ECC Workshop
"Can we obtain higher speeds at comparable security levels using genus-2 hyperelliptic curves?"
Unfortunately:
"genus-2 point counting is too slow to reach 256 bits"

No point counting → no cryptographic genus 2 curves

Flashback?!



Wasn't this considered before?

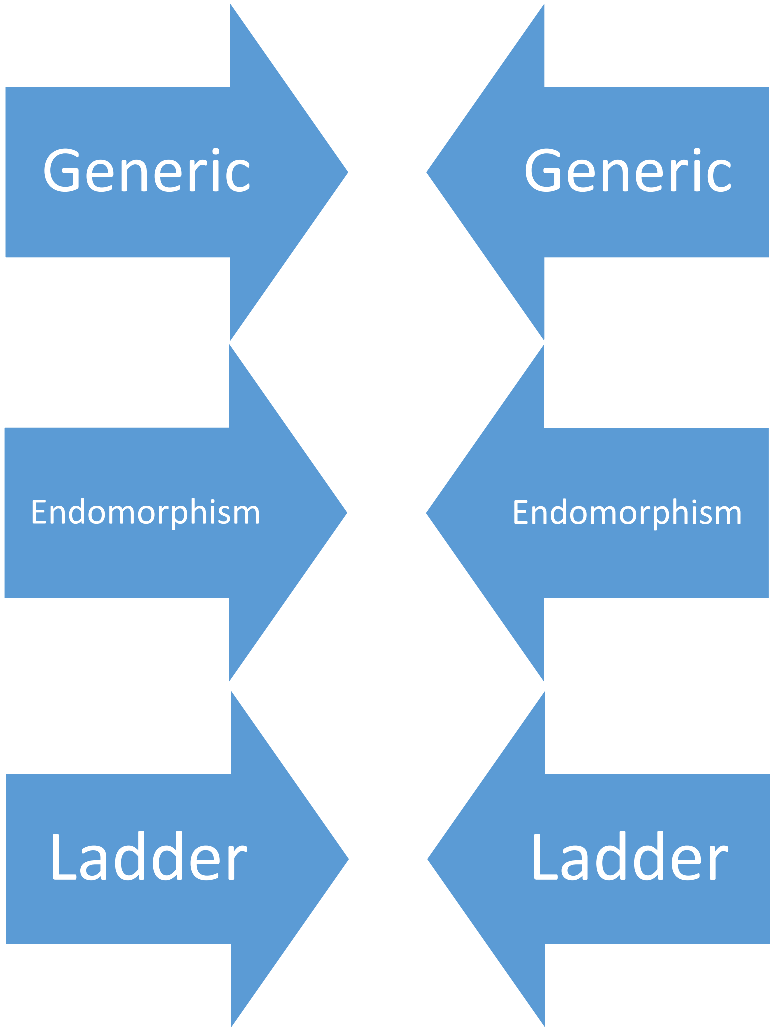
2006: D. J. Bernstein: *Elliptic vs. hyperelliptic*, ECC Workshop
"Can we obtain higher speeds at comparable security levels using genus-2 hyperelliptic curves?"
Unfortunately:
"genus-2 point counting is too slow to reach 256 bits"

No point counting → no cryptographic genus 2 curves

Fortunately, there has been significant progress
2011: Gaudry-Kohel-Smith: *Counting points on genus 2 curves with real multiplication*, Asiacrypt
2012: Gaudry-Schost: *Genus 2 point counting over prime fields*, J. Symb. Comput.

After seven years Genus 2 is ready to rumble!

Genus 1 versus Genus 2

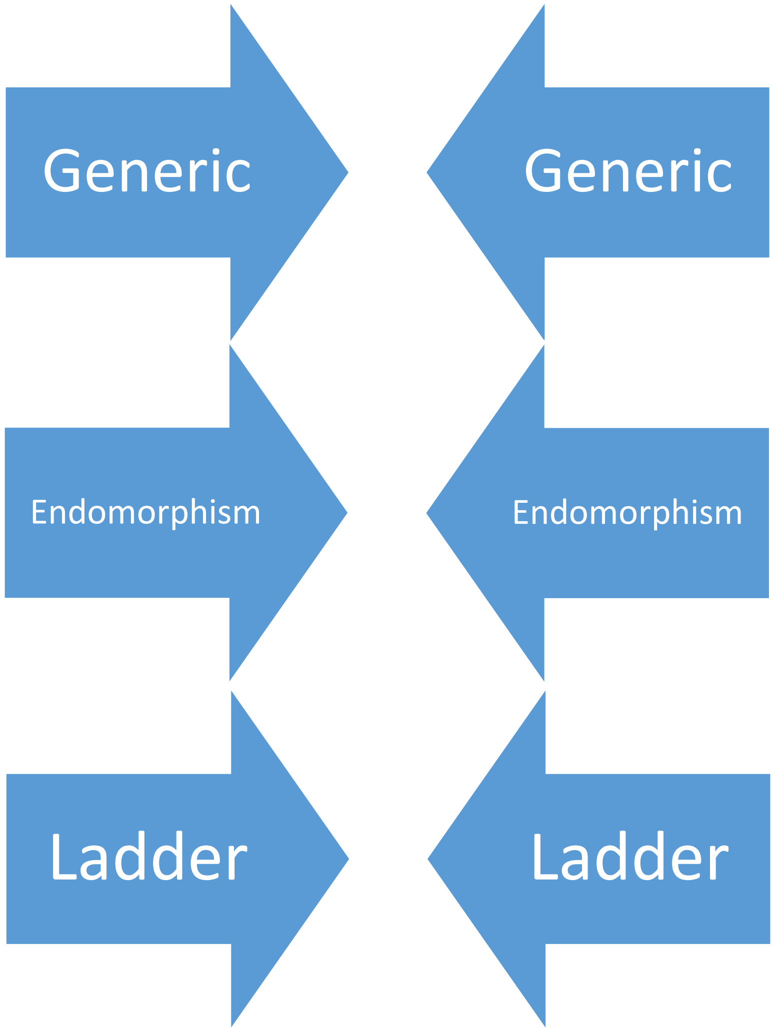


Practical performance comparison
Genus 1 versus Genus 2

- 128-bit security level
- High-end 64-bit platforms
  (although we considered embedded devices as well)
- Use all the available tricks!

After seven years Genus 2 is ready to rumble!

Genus 1 versus Genus 2



Practical performance comparison
Genus 1 versus Genus 2

- 128-bit security level
- High-end 64-bit platforms
  (although we considered embedded devices as well)
- Use all the available tricks!

- Let’s start with an arithmetic interlude:
  Why do we care about “special” primes?

Mersenne to the rescue!

In genus 1 “special” primes are used
to speed-up modular reduction

- NIST $p_{224} = 2^{224} - 2^{96} + 1$
- NIST $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- Bernstein $p_{25519} = 2^{255} - 19$

Mersenne to the rescue!

In genus 1 “special” primes are used to speed-up modular reduction

- NIST $p_{224} = 2^{224} - 2^{96} + 1$
- NIST $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- Bernstein $p_{25519} = 2^{255} - 19$

Mersenne primes

- Prime of the form $2^q - 1$, with q prime
- Allows **very** efficient modular arithmetic
- Gaudry-Schost found a cryptographic Kummer surface over $\mathbf{F}_p$ with $p = 2^{127} - 1$

#	q
1	2
2	3
3	5
4	7
5	13
6	17
7	19
8	31
9	61
10	89
11	107
12	127
13	521
14	607

Mersenne to the rescue!

In genus 1 “special” primes are used to speed-up modular reduction

- NIST $p_{224} = 2^{224} - 2^{96} + 1$
- NIST $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- Bernstein $p_{25519} = 2^{255} - 19$

Mersenne primes

- Prime of the form $2^q - 1$, with q prime
- Allows **very** efficient modular arithmetic
- Gaudry-Schost found a cryptographic Kummer surface over $\mathbf{F}_p$ with $p = 2^{127} - 1$

#	q
1	2
2	3
3	5
4	7
5	13
6	17
7	19
8	31
9	61
10	89
11	107
12	127
13	521
14	607

$\approx$ 128-bit security for genus 2

NIST-p521

Mersenne to the rescue! – Modular addition

$a + b < 2^{128}$

Zero is represented by
0 or $2^{127} - 1$

$$c = a + b \bmod (2^{127}-1) = \begin{cases} a + b & \text{if } (a+b) \leq 2^{127} - 1 \\ a + b - (2^{127}-1) & \text{if } (a+b) > 2^{127} - 1 \end{cases}$$

Constant-time: addition + conditional subtraction

= addition +subtraction + masking (uses registers)

Mersenne to the rescue! – Modular addition

$a + b < 2^{128}$

Zero is represented by
0 or $2^{127} - 1$

$$c = a + b \bmod (2^{127}-1) = \begin{cases} a + b & \text{if } (a+b) \leq 2^{127} - 1 \\ a + b - (2^{127}-1) & \text{if } (a+b) > 2^{127} - 1 \end{cases}$$

Constant-time: addition + conditional subtraction

= addition +subtraction + masking (uses registers)

$$R(x) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor (2^{127}-1) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor 2^{127} + \left\lfloor \frac{x}{2^{127}} \right\rfloor$$

Mersenne to the rescue! – Modular addition

$a + b < 2^{128}$

$$c = a + b \bmod (2^{127} - 1) = \begin{cases} a + b & \text{if } (a+b) \leq 2^{127} - 1 \\ a + b - (2^{127} - 1) & \text{if } (a+b) > 2^{127} - 1 \end{cases}$$

Constant-time: addition + conditional subtraction

= addition +subtraction + masking (uses registers)

$$R(x) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor (2^{127} - 1) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor 2^{127} + \left\lfloor \frac{x}{2^{127}} \right\rfloor$$

If the **msb is zero** then leave it at **zero**
If the **msb is one** then set it to **zero**
Idea: use the **bit-reset** instruction!

$\in \{0, 1\}$

Mersenne to the rescue! – Modular addition

$a + b < 2^{128}$

$$c = a + b \bmod (2^{127} - 1) = \begin{cases} a + b & \text{if } (a+b) \leq 2^{127} - 1 \\ a + b - (2^{127} - 1) & \text{if } (a+b) > 2^{127} - 1 \end{cases}$$

Constant-time: addition + conditional subtraction

= addition +subtraction + masking (uses registers)

$$R(x) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor (2^{127} - 1) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor 2^{127} + \left\lfloor \frac{x}{2^{127}} \right\rfloor$$

Compute: $c = R(a+b)$ when $0 \leq a, b < 2^{127}$ then $0 \leq c < 2^{127}$

Avoid masking and extra register usage

Cost modular addition: 2x `add` + 1x `bit-reset` instruction

Mersenne to the rescue! – Modular multiplication

$c = a \times b = c_H 2^{128} + c_L$, with

$0 \le a, b < 2^{127}, 0 \le c_L < 2^{128}$ and $0 < c_H \le \left\lfloor \frac{\left(2^{127}-1\right)^2}{2^{128}} \right\rfloor = 2^{126} - 1$

$$c \equiv c_H 2^{128} + c_L - 2c_H\,(2^{127} - 1) \equiv c_L + 2c_H \left(\mathrm{mod}\ (2^{127} - 1)\right)$$

Mersenne to the rescue! – Modular multiplication

$c = a \times b = c_H 2^{128} + c_L$, with

$0 \le a, b < 2^{127}, 0 \le c_L < 2^{128}$ and $0 < c_H \le \left\lfloor \frac{(2^{127}-1)^2}{2^{128}} \right\rfloor = 2^{126} - 1$

$$c \equiv c_H 2^{128} + c_L - 2c_H\,(2^{127} - 1) \equiv c_L + 2c_H \pmod{(2^{127} - 1)}$$

Can be $> 2^{128}$

$$c \equiv R(R(c_L) + 2c_H) \pmod{(2^{127} - 1)}$$

Mersenne to the rescue! – Modular multiplication

$c = a \times b = c_H 2^{128} + c_L$, with

$0 \le a, b < 2^{127}, 0 \le c_L < 2^{128}$ and $0 < c_H \le \left\lfloor \frac{(2^{127}-1)^2}{2^{128}} \right\rfloor = 2^{126} - 1$

$$c \equiv c_H 2^{128} + c_L - 2c_H\,(2^{127} - 1) \equiv c_L + 2c_H \pmod{(2^{127} - 1)}$$

Can be $> 2^{128}$

$$c \equiv R(R(c_L) + 2c_H) \pmod{(2^{127} - 1)}$$

$\le 2^{127} + 2(2^{126} - 1)$
$= 2(2^{127} - 1)$

Reduction cost: 6x `add`, 2x `bit-reset`, 1x `shift`

Mersenne to the rescue! – Modular multiplication

$c = a \times b = c_H 2^{128} + c_L$, with

$0 \le a, b < 2^{127}, 0 \le c_L < 2^{128}$ and $0 < c_H \le \left\lfloor \frac{(2^{127}-1)^2}{2^{128}} \right\rfloor = 2^{126} - 1$

$$c \equiv c_H 2^{128} + c_L - 2c_H\,(2^{127} - 1) \equiv c_L + 2c_H \pmod{(2^{127} - 1)}$$

Can be $> 2^{128}$

$$c \equiv R(R(c_L) + 2c_H) \pmod{(2^{127} - 1)}$$

$\le 2^{127} + 2(2^{126} - 1)$
$= 2(2^{127} - 1)$

Reduction cost: 6x `add`, 2x `bit-reset`, 1x `shift`
Multiplication: 4x `mul` and 5x `add` instruction

Montgomery friendly primes

Interleaved radix-2^b Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

C=0

for $i = 0$ to $n - 1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C + q \cdot p}{2^b}$$

Montgomery: *Modular Multiplication Without Trial Division*. Math. of Comp. 1985

Montgomery friendly primes

Interleaved radix-2^b Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

C=0

for $i = 0$ to $n - 1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C + q \cdot p}{2^b}$$

Not much we can do: this is the multiplication

Montgomery: *Modular Multiplication Without Trial Division*. Math. of Comp. 1985

Montgomery friendly primes

Interleaved radix-2^b Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

C=0

for $i = 0$ to $n - 1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C+q \cdot p}{2^b}$$

Not much we can do: this is the multiplication

If $p = \pm 1 \bmod 2^{\mathrm{b}}$ then $\mu = \mp 1 \bmod 2^{\mathrm{b}}$

Montgomery: *Modular Multiplication Without Trial Division*. Math. of Comp. 1985

Montgomery friendly primes

Interleaved radix-2^b Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

C=0

for $i = 0$ to $n - 1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C + q \cdot p}{2^b}$$

Not much we can do: this is the multiplication

If $p = \pm 1 \bmod 2^{\mathrm{b}}$ then $\mu = \mp 1 \bmod 2^{\mathrm{b}}$

Additionally, if *p* has a “special” form: avoid muls

Montgomery: *Modular Multiplication Without Trial Division*. Math. of Comp. 1985

Montgomery friendly primes

Interleaved radix-2^b Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \bmod p, \mu = -p^{-1} \bmod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

C=0

for $i = 0$ to $n - 1$ do

$$C = C + a_i \cdot B$$
$$q = \mu \cdot C \bmod 2^b$$
$$C = \frac{C + q \cdot p}{2^b}$$

Not much we can do: this is the multiplication

If $p = \pm 1 \bmod 2^{\mathrm{b}}$ then $\mu = \mp 1 \bmod 2^{\mathrm{b}}$

Additionally, if *p* has a “special” form: avoid muls

Example: $2^b\left(2^{\tilde{b}} - c\right) - 1$

$$2^{127} - 1 = 2^{64}(2^{63} - 0) - 1$$

Montgomery: *Modular Multiplication Without Trial Division*. Math. of Comp. 1985

Security & Benchmark Platform

Benchmark Platform

- Intel Core i7-3520M (Ivy Bridge) processor at 2893.484 MHz
- hyperthreading turned off and over-clocking (“turbo boost”) disabled

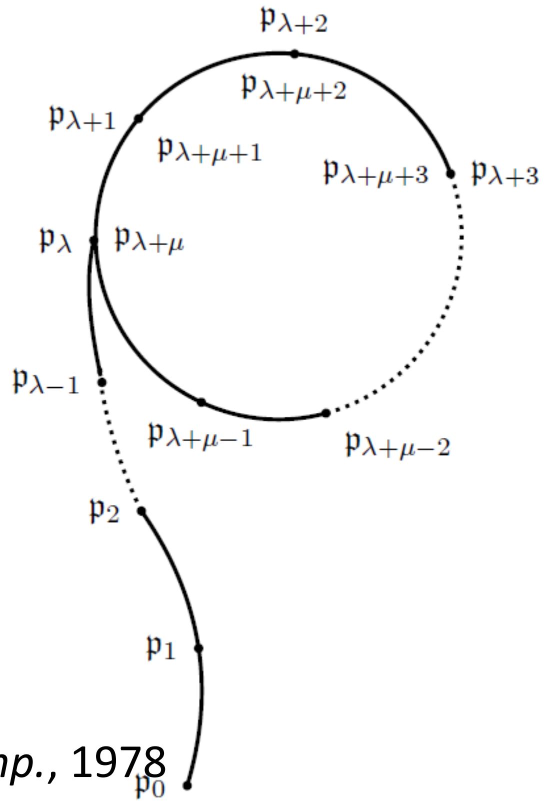
Security & Benchmark Platform

Benchmark Platform

- Intel Core i7-3520M (Ivy Bridge) processor at 2893.484 MHz
- hyperthreading turned off and over-clocking (“turbo boost”) disabled

Generic Attack: Pollard rho

- [Pollard-MoC78]
- $\sqrt{(\pi r)/(2\#\mathrm{Aut})}$, where $\#\mathrm{Aut} \geq 2$ for curves with group order $h \cdot r$



J. M. Pollard: Monte Carlo methods for index computation (mod p). *Math. Comp.*, 1978

Battle #1

NISTp-256 versus Generic1271

Battle #1

NISTp-256 versus Generic1271

Generic genus 1 versus Generic genus 2

Generic?

- No special requirements on the curve
- Techniques can be applied to **all** genus 1 or genus 2 curves
- Use “special” primes for efficiency
- Use prime order curves for optimal security

NISTp-256 versus Generic1271

	NISTp-256	Generic1271
p	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	$\begin{cases} 2^{127} - 1 & (\boldsymbol{a}) \\ 2^{64}(2^{63} - 0) - 1 & (\boldsymbol{b}) \end{cases}$
Order	Prime order	Prime order
Scalar multiplication	windowing	windowing
Coordinate / curve	Jacobian coordinates with $a = -3$ for short Weierstrass curves	[CL]
Security	$\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{127.8}$	$\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{126.8}$

We use arithmetic on imaginary quadratic curves using homogeneous projective coordinates.
We optimized the formulas from:

[CL] Costello, Lauter: *Group law computations on Jacobians of hyperelliptic curves*. SAC 2011

NISTp-256 versus Generic1271

	NISTp-256	Generic1271
p	$2^{256}-2^{224}+2^{192}+2^{96}-1$	$\begin{cases} 2^{127}-1 & (\boldsymbol{a}) \\ 2^{64}(2^{63}-0)-1 & (\boldsymbol{b}) \end{cases}$
Order	Prime order	Prime order
Scalar multiplication	windowing	windowing
Coordinate / curve	Jacobian coordinates with $a=-3$ for short Weierstrass curves	[CL]
Security	$\sqrt{(\pi r)/(2\cdot 2)} \approx 2^{127.8}$	$\sqrt{(\pi r)/(2\cdot 2)} \approx 2^{126.8}$
Double	3**M**+5**S**	34**M**+6**S**
Addition	11**M**+5**S**	44**M**+4**S**
Mixed addition	7**M**+4**S**	37**M**+5**S**

NISTp-256 versus Generic1271

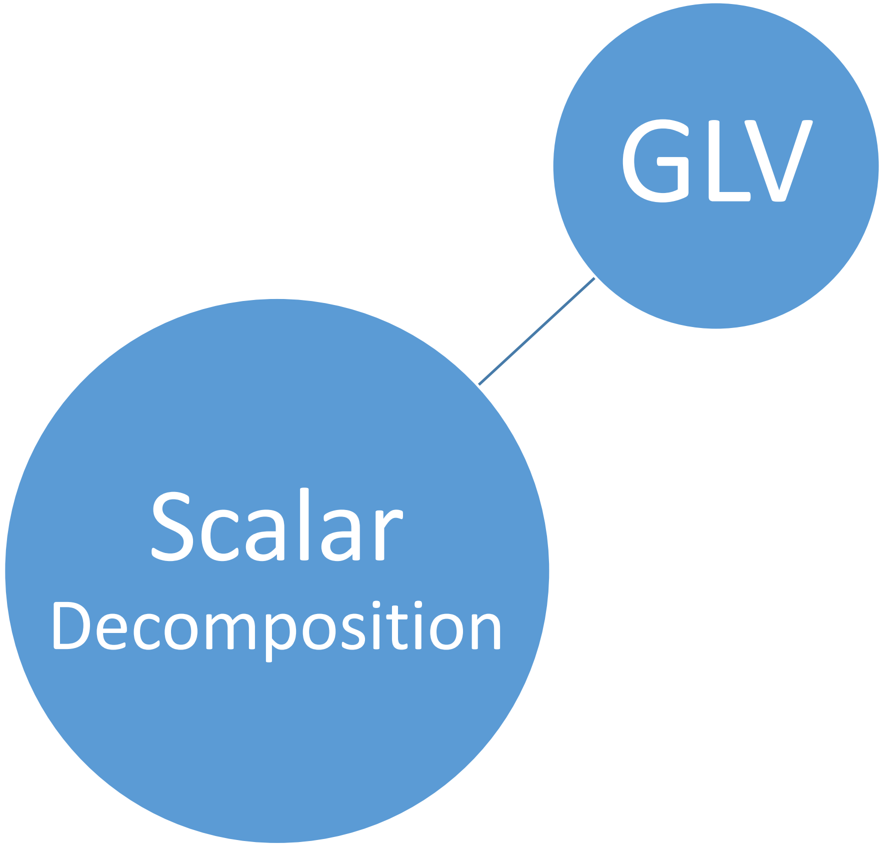
	NISTp-256	Generic1271
p	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	$\begin{cases} 2^{127} - 1 & (\boldsymbol{a}) \\ 2^{64}(2^{63} - 0) - 1 & (\boldsymbol{b}) \end{cases}$
Order	Prime order	Prime order
Scalar multiplication	windowing	windowing
Coordinate / curve	Jacobian coordinates with $a = -3$ for short Weierstrass curves	[CL]
Security	$\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{127.8}$	$\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{126.8}$

Genus 1: NISTp-256	658,000
Genus 2: generic1271 (a)	248,000
Genus 2: generic1271 (b)	295,000

Battle #2

GLV-j=0 versus BuhlerKoblitzGLV

Scalar Decomposition over Prime Fields



- Gallant, Lambert, Vanstone [GLV-C01]
- Use non-trivial endomorphism
- Larger endomorphism ring means larger dimensional scalar decomposition

Scalar Decomposition over Prime Fields

Scalar Decomposition

GLV

- Gallant, Lambert, Vanstone [GLV-C01]
- Use non-trivial endomorphism
- Larger endomorphism ring means larger dimensional scalar decomposition

Genus 1 over $\mathbf{F}_p$
256-bit primes
Allows: 2-GLV

Genus 2 over $\mathbf{F}_p$
128-bit primes
Allows: 4-GLV

Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* “mini-scalars” $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\left([\lambda^{d-1}]P\right)$$

Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\big([\lambda^{d-1}]P\big)$$

Approach #1

$k_0 = \boxed{k_{0,0}}\ \boxed{k_{0,1}}\ \boxed{k_{0,2}}\ \boxed{k_{0,3}}$

$k_1 = \boxed{k_{1,0}}\ \boxed{k_{1,1}}\ \boxed{k_{1,2}}\ \boxed{k_{1,3}}$

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i\lambda^i]\,P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\big([\lambda^{d-1}]P\big)$$

Approach #1

$$k_0 = \boxed{k_{0,0}}\ \boxed{k_{0,1}}\ \boxed{k_{0,2}}\ \boxed{k_{0,3}}$$

$$k_1 = \boxed{k_{1,0}}\ \boxed{k_{1,1}}\ \boxed{k_{1,2}}\ \boxed{k_{1,3}}$$

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\left([\lambda^{d-1}]P\right)$$

Approach #1

$k_0 =$	$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_1 =$	$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* “mini-scalars” $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\big([\lambda^{d-1}]P\big)$$

Approach #1

$k_0 =$	$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_1 =$	$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* "mini-scalars" $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\left([\lambda^{d-1}]P\right)$$

Approach #1

$k_0 =$	$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_1 =$	$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

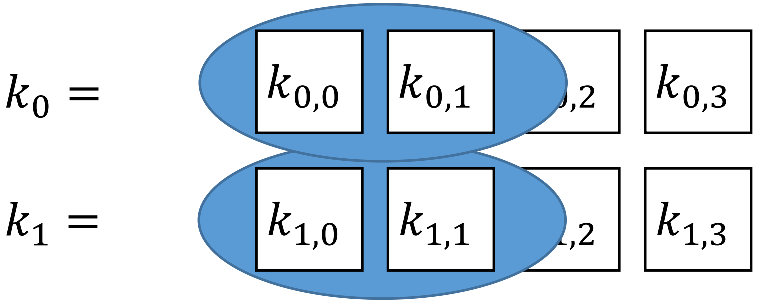
Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* “mini-scalars” $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\left([\lambda^{d-1}]P\right)$$

Approach #2



Precompute: $\begin{cases} \{\emptyset, P, 2P, 3P\} \\ \{\emptyset, [\lambda]P, 2[\lambda]P, 3[\lambda]P\} \end{cases}$

Example: $d = 2$

Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar *k* into *d* “mini-scalars” $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these *d* smaller scalars

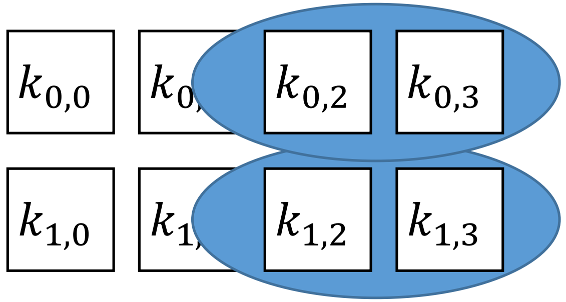
Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]\, P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]\left([\lambda^{d-1}]P\right)$$

Approach #2

$k_0 =$

$k_1 =$



Precompute: $\begin{cases} \{\emptyset, P, 2P, 3P\} \\ \{\emptyset, [\lambda]P, 2[\lambda]P, 3[\lambda]P\} \end{cases}$

Example: $d = 2$

BuhlerKoblitzGLV – 4-dimensional GLV

Buhler-Koblitz curves

- $C/\mathbf{F}_p : \; y^2 = x^5 + a$
- ψ: $\mathrm{Jac}(C) \to \mathrm{Jac}(C)$,
  $\psi(D) = [\lambda]D$, for $0 < \lambda < r$
- Decompose the scalar using [PJL]
  Cost: 20 long integer muls

Curve Choice

$$\begin{cases} p_{127m} = (2^{63} - 27433)2^{64} + 1 \\ a = 17 \end{cases}$$

$$\mu = -p_{127m}^{-1} \bmod 2^{64} = -1$$

254-bit prime order

$$\begin{cases} p_{128n} = 2^{128} - 24935 \\ a = 3^7 \end{cases}$$

256-bit prime order

[PJL] Park, Jeong, Lim: *Speeding up point multiplication on hyperelliptic curves with efficiently-computable endomorphisms*. Eurocrypt 2002

BuhlerKoblitzGLV – 4-dimensional GLV

Offline

Pre-compute 2^4 points

11**A**+3$\boldsymbol{\psi}$

Online

64**D**+64**A**

Curve Choice

$$\begin{cases} p_{127m} = (2^{63} - 27433)2^{64} + 1 \\ a = 17 \end{cases}$$

$$\mu = -p_{127m}^{-1} \bmod 2^{64} = -1$$

254-bit prime order

$$\begin{cases} p_{128n} = 2^{128} - 24935 \\ a = 3^7 \end{cases}$$

256-bit prime order

BuhlerKoblitzGLV – 4-dimensional GLV

Offline

Pre-compute 2^4 points

11**A**+3ψ+1**I**+(3+4)·15**M**

Online

64**D**+64**A** → 64**D**+64**MA**

Recall: A=44**M**+4**S**, MA=37**M**+5**S**

Additional cost: 1**I**+105**M**

Savings: 64(**A**-**MA**)=448**M**-64**S**

Speedup when: **I** < 279**M**

Curve Choice

$$\begin{cases} p_{127m} = (2^{63} - 27433)2^{64} + 1 \\ a = 17 \end{cases}$$

$$\mu = -p_{127m}^{-1} \bmod 2^{64} = -1$$

254-bit prime order

$$\begin{cases} p_{128n} = 2^{128} - 24935 \\ a = 3^7 \end{cases}$$

256-bit prime order

Montgomery: *Speeding the Pollard and elliptic curve methods of factorization. Math. of Comp. 1987*

GLV-j=0 versus BuhlerKoblitzGLV

	GLV-j=0	BuhlerKoblitzGLV
p	$2^{256} - 11733$	$\begin{cases} 2^{128} - 24935 & (\boldsymbol{a}) \\ (2^{63} - 27433)2^{64} + 1 & (\boldsymbol{b}) \end{cases}$
Order	Prime order	Prime order
Scalar multiplication	2-dimensional GLV	4-dimensional GLV (approach #1)
Coordinate / curve	j-invariant 0 in Weierstrass form $y^2 = x^3 + 2$	Buhler-Koblitz curve $y^2 = x^5 + a$

Longa, Sica: *Four-dimensional Gallant-Lambert-Vanstone scalar multiplication*. Asiacrypt 2012

GLV-j=0 versus BuhlerKoblitzGLV

	GLV-j=0	BuhlerKoblitzGLV
p	$2^{256} - 11733$	$\begin{cases} 2^{128} - 24935 & (\boldsymbol{a}) \\ (2^{63} - 27433)2^{64} + 1 & (\boldsymbol{b}) \end{cases}$
Order	Prime order	Prime order
Scalar multiplication	2-dimensional GLV	4-dimensional GLV (approach #1)
Cost scalar multiplication	1**I** + 904**M** + 690**S**	20 integer muls + 3$\boldsymbol{\psi}$+2**I**+5005**M**+748**S**
Security	$\sqrt{(\pi r) / (2 \cdot 6)} \approx 2^{127.0}$	$\sqrt{(\pi r) / (2 \cdot 10)} \approx 2^{125.7}$

Longa, Sica: *Four-dimensional Gallant-Lambert-Vanstone scalar multiplication*. Asiacrypt 2012

GLV-j=0 versus BuhlerKoblitzGLV

	GLV-j=0	BuhlerKoblitzGLV
p	$2^{256} - 11733$	$\begin{cases} 2^{128} - 24935 & (\boldsymbol{a}) \\ (2^{63} - 27433)2^{64} + 1 & (\boldsymbol{b}) \end{cases}$
Order	Prime order	Prime order
Scalar multiplication	2-dimensional GLV	4-dimensional GLV (approach #1)
Cost scalar multiplication	1**I** + 904**M** + 690**S**	20 integer muls + 3$\boldsymbol{\psi}$+2**I**+5005**M**+748**S**
Security	$\sqrt{(\pi r)/(2 \cdot 6)} \approx 2^{127.0}$	$\sqrt{(\pi r)/(2 \cdot 10)} \approx 2^{125.7}$

Genus 1: GLV-j=0	145,000
Genus 2: BuhlerKoblitzGLV (a)	164,000
Genus 2: BuhlerKoblitzGLV (b)	156,000

Longa, Sica: *Four-dimensional Gallant-Lambert-Vanstone scalar multiplication*. Asiacrypt 2012

Battle #3

curve25519 versus Kummer1271

Battle #3

curve25519 versus Kummer1271

Use the Kummer surface from
Gaudry, Schost: *Genus 2 point counting over prime fields*, J. Symb. Comput., 2012

Laddering algorithms

Elliptic curves

- [M] differential addition: compute $P + Q$ from $\{P, Q, P - Q\}$ without y-coord
- to compute kP keep $\{mP, (m + 1)P\}$ such that $(m + 1)P - mP = P$
- Identify $P = (P_x, P_y)$ and $-P = (P_x, -P_y)$
- Cost for double+differential add: 5**M** + 4**S**

[M] Montgomery: *Speeding the Pollard and elliptic curve methods of factorization*. Math. of Comp. 1987

Laddering algorithms

Elliptic curves

- [M] differential addition: compute $P + Q$ from $\{P, Q, P - Q\}$ without y-coord
- to compute kP keep $\{mP, (m+1)P\}$ such that $(m+1)P - mP = P$
- Identify $P = (P_x, P_y)$ and $-P = (P_x, -P_y)$
- Cost for double+differential add: 5**M** + 4**S**

Genus 2 curves

Work on the Kummer surface associated to a Jacobian, rather than on the Jacobian itself

- [SS] genus 2 analogue $\mathrm{Jac}(C) \to K$ is 2-to-1
- [G] faster Kummer surface
- [C] even faster "squares only" setting on the Kummer surface
- Cost for double+differential add: 16**M** + 9**S**

[M] Montgomery: *Speeding the Pollard and elliptic curve methods of factorization*. Math. of Comp. 1987

[SS] Smart, Siksek: *A fast Diffie-Hellman protocol in genus 2*. J. of Cryptology. 1999

[G] Gaudry: *Fast genus 2 arithmetic based on theta functions*. J. of Math. Cryptology. 2007

[C] Cosset: *Factorization with genus 2 curves*. Math. of Comp. 2010

Laddering algorithms

Elliptic curves

- [M] differential addition: compute $P + Q$ from $\{P, Q, P - Q\}$ without y-coord
- to compute kP keep $\{mP, (m + 1)P\}$ such that $(m + 1)P - mP = P$
- Identify $P = (P_x, P_y)$ and $-P = (P_x, -P_y)$
- Cost for double+differential add: 5**M** + 4**S**

Genus 2 curves

Work on the Kummer surface associated to a Jacobian, rather than on the Jacobian itself

- [SS] genus 2 analogue $\mathrm{Jac}(C) \to K$ is 2-to-1
- [G] faster Kummer surface
- [C] even faster "squares only" setting on the Kummer surface
- Cost for double+differential add: 16**M** + 9**S**

- no additions: does allow scalar multiplication
- attractive setting for Diffie-Hellman like protocols
- Inherently runs in constant time

[M] Montgomery: *Speeding the Pollard and elliptic curve methods of factorization*. Math. of Comp. 1987
[SS] Smart, Siksek: *A fast Diffie-Hellman protocol in genus 2*. J. of Cryptology. 1999
[G] Gaudry: *Fast genus 2 arithmetic based on theta functions*. J. of Math. Cryptology. 2007
[C] Cosset: *Factorization with genus 2 curves*. Math. of Comp. 2010

curve25519 versus Kummer1271

	curve25519	Kummer1271
p	$2^{255} - 19$	$\begin{cases} 2^{127} - 1 & (\boldsymbol{a}) \\ 2^{64}(2^{63} - 0) - 1 & (\boldsymbol{b}) \end{cases}$
Order	$8 \cdot 253$-bit prime / $4 \cdot 253$-bit prime	$16 \cdot 250$-bit prime / $16 \cdot 251$-bit prime
Scalar multiplication	Montgomery ladder	Kummer ladder
Coordinate / curve	Montgomery curve	"Squares only" setting on a Kummer surface
Double + dif. add	5**M** + 4**S**	16**M** + 9**S**
Security	$\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{125.8}$	$\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{124.8}$

Bernstein: *Curve25519: New Diffie-Hellman speed records.* PKC 2006

Bernstein, Duif, Lange, Schwabe: *High-speed high-security signatures*. CHES 2011

curve25519 versus Kummer1271

	curve25519	Kummer1271
p	$2^{255} - 19$	$\begin{cases} 2^{127} - 1 & (\boldsymbol{a}) \\ 2^{64}(2^{63} - 0) - 1 & (\boldsymbol{b}) \end{cases}$
Order	$8 \cdot 253$-bit prime / $4 \cdot 253$-bit prime	$16 \cdot 250$-bit prime / $16 \cdot 251$-bit prime
Scalar multiplication	Montgomery ladder	Kummer ladder
Double + dif. add	5**M** + 4**S**	16**M** + 9**S**
Security	$\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{125.8}$	$\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{124.8}$

Genus 1: curve25519	182,000
Genus 2: Kummer1271 (a)	117,000
Genus 2: Kummer1271 (b)	139,000

Bernstein: *Curve25519: New Diffie-Hellman speed records.* PKC 2006

Bernstein, Duif, Lange, Schwabe: *High-speed high-security signatures*. CHES 2011

Summary: genus 1 versus genus 2 over prime fields

Curve	cycles	CT	protocols
Genus 1: NISTp-256	658,000	?	all
Genus 2: generic1271 (a)	248,000	☒	all
Genus 1: GLV-j=0	145,000	☒	all
Genus 2: BuhlerKoblitzGLV (b)	156,000	☒	all
Genus 1: curve25519	182,000	☑	some
Genus 2: Kummer1271 (a)	117,000	☑	some

Generic

- Genus 2 $>$ 2.5 faster than genus 1
- Mersenne prime $2^{127} - 1$ **very** efficient in practice
- NISTp-256 arithmetic $(2^{256} - 2^{224} + 2^{192} + 2^{96} - 1)$ is relatively slow

Summary: genus 1 versus genus 2 over prime fields

Curve	cycles	CT	protocols
Genus 1: NISTp-256	658,000	?	all
Genus 2: generic1271 (a)	248,000	☒	all
Genus 1: GLV-j=0	145,000	☒	all
Genus 2: BuhlerKoblitzGLV (b)	156,000	☒	all
Genus 1: curve25519	182,000	☑	some
Genus 2: Kummer1271 (a)	117,000	☑	some

Endomorphism

- Genus 1 slightly faster than genus 2
  (better genus 1 assembly implementation?)
- Montgomery friendly primes **faster** than primes of the form $2^{128} - c$

Summary: genus 1 versus genus 2 over prime fields

Curve	cycles	CT	protocols
Genus 1: NISTp-256	658,000	?	all
Genus 2: generic1271 (a)	248,000	☒	all
Genus 1: GLV-j=0	145,000	☒	all
Genus 2: BuhlerKoblitzGLV (b)	156,000	☒	all
Genus 1: curve25519	182,000	☑	some
Genus 2: Kummer1271 (a)	117,000	☑	some

Ladder

- Genus 2 faster than genus 1
- Thanks to the Kummer surface by Gaudry & Schost
  the Mersenne prime $2^{127}-1$ comes to the rescue again

Conclusions

Genus 2 has many advantages over elliptic curves

- ✓ Larger endomorphism ring
  4-GLV possible in genus 2 versus 2-GLV in genus 1
- ✓ Can use the Mersenne prime $2^{127} - 1$
- ✓ Laddering using the Kummer surface is very efficient
- ✓ This results are on a 64-bit platform, smaller primes have more potential on embedded devices

Final score

genus 1 *versus* genus 2

1 : **2**

Conclusions

Related / ongoing work

- Genus 2 curves over $\mathbf{F}_{p^2}$ $\rightarrow$ 8-dimensional scalar decomposition
  Allows for 64-bit primes p
  Faster attacks, reduced security from 128-bit to ≈112-bit
- Practical analysis of security genus 1 versus genus 2 over $\mathbf{F}_p$
  What is the effect of using the automorphism group in practice?

Future work

- Unlikely to attract attention from industry if less than order of magnitude faster:
  More work is needed!
- Using endomorphisms on the Kummer surface?

Conclusions

Use elliptic or genus 2 curves?

Related / ongoing work

- Genus 2 curves over $\mathbf{F}_{p^2}$ → 8-dimensional scalar decomposition
  Allows for 64-bit primes p
  Faster attacks, reduced security from 128-bit to ≈112-bit
- Practical analysis of security genus 1 versus genus 2 over $\mathbf{F}_p$
  What is the effect of using the automorphism group in practice?

Future work

- Unlikely to attract attention from industry if less than order of magnitude faster:
  More work is needed!
- Using endomorphisms on the Kummer surface?

Conclusions

Use elliptic or genus 2 curves?

Related / ongoing work

- Genus 2 curves over $\mathbf{F}_{p^2}$ → 8-dimensional scalar decomposition
  Allows for 64-bit primes p
  Faster attacks, reduced security from 128-bit to ≈112-bit
- Practical analysis of security genus 1 versus genus 2 over $\mathbf{F}_p$
  What is the effect of using the automorphism group in practice?

Future work

- Unlikely to attract attention from industry if less than order of magnitude faster:
  More work is needed!
- Using endomorphisms on the Kummer surface?

Difficult to see. Always in motion is the future.
YODA, *Star Wars Episode V: The Empire Strikes Back*